Athens University of Economics and Business Dept. of Informatics
M.Sc. Program in Data Science
INF322: Social Network Analysis Fall 2019
Instructor: Katia Papakonstantinopoulou

Homework 1: SNAP

High performance system for analysis and manipulation of large networks

Due 11:59pm EEST November 25, 2019

General Instructions

Your answers should be as concise as possible.

Submission instructions: You should submit your answers and code in a compressed directory uploaded via `https://eclass.aueb.gr`

Submitting answers: Prepare answers to your homework in a single PDF file named *hw1.pdf*

Submitting code: Prepare your two files of code (one for each problem) named *hw1-1.py* and *hw1-2.py*, respectively.

Problems

SNAP (Stanford Network Analysis Project) is a C++ based general purpose, high performance system for analysis and manipulation of large networks. SNAP scales to massive networks with hundreds of millions of nodes, and billions of edges. You can use it for a variety of graph related tasks, such as calculating structural properties, generating graphs and applying graph algorithms.

In this homework you will be using Snap.py, a Python interface for SNAP. The Snap.py interface and everything you will need for this homework is preinstalled in the Virtual Machine provided by this course.[1] Alternatively, you can proceed with installing SNAP to your own platform through this link: `https://snap.stanford.edu/snappy/#download`.

An introductory script for Snap.py use can be found here:
`https://snap.stanford.edu/snappy/file/intro.py`

[1]`https://goo.gl/CGm1YR`

1 Euler Paths and Circuits

The first part of this homework requires that you familiarize yourselves with *graph creation* and *graph traversal* with SNAP.

You have to develop two Python functions that examine whether a given graph has:

- an Euler Path, and
- an Euler Circuit.

An Euler path is a path that uses every edge of a graph exactly once. An Euler path starts and ends at different vertices. If a graph has an Euler path, then it must have exactly two vertices with odd degree, and it is these odd vertices that will form the beginning and end of the path.

An Euler circuit is a circuit that uses every edge of a graph exactly once. An Euler circuit starts and ends at the same vertex. If a graph has an Euler circuit, then all of its vertices must be of even degree.

Note that you also must verify that the graph is connected for both cases, which can be easily done using SNAP:
`https://snap.stanford.edu/snappy/doc/reference/cncom.html`.

In addition to implementing the above functions you should complete the test-case that is provided with this homework (*hw1-1.py*). In particular, you should complete all four tests by filling in code that creates graphs that satisfy the tests' assertions:

- A graph that has an Euler path (but not an Euler circuit),
- A graph that does not have an Euler path,
- A graph that has an Euler circuit, and
- A graph that does not have an Euler circuit.

Note that submitting your homework with tests that succeed does not guarantee full points in this exercise. You should make sure that both functions are correctly implemented, and the graphs in your tests are created appropriately. Remember that if you wish to create a graph of significant size, you can employ one of the available graph generators:
`https://snap.stanford.edu/snappy/doc/reference/generators.html`.

2 Apply node centrality measures and community detection algorithms on generated graphs

For the second part of this homework, you will have to write a Python script (*hw1-2.py*) that generates a graph of given size, reports some information on the graph, and compares the execution times of two community detection algorithms.

You will generate graphs using the *Watts-Strogatz* model. Start with a graph of 50 nodes and set the out-degree of each node to a value of your choice in $[5, 20]$. First, your script will print out the id of the node with the highest degree as well as its degree. Then, you will print out the ids of the nodes with the highest *Hub* and *Authority* scores as well as their scores. Finally, you will measure the time needed for the execution of the *Girvan-Newman* community detection algorithm based on betweenness centrality and the *Clauset-Newman-Moore* community detection method.

Your task is to execute this script multiple times by increasing the number of nodes parameter, to report the execution time for graphs of different sizes. You should repeat execution to the point that both algorithms require more than 10 minutes to execute or you receive a memory error.

Finally, for the largest among the graphs you will generate, find the top-30 nodes of highest PageRank and compute the following centrality measures for them: Betweenness, Closeness, Authority score and Hub score. Then compare the following groups of metrics, using one plot for each group:

- Betweenness, Closeness and PageRank.
- PageRank, Authority score and Hub score.

Each plot should illustrate the corresponding measures for the 30 nodes ranked by decreasing order of their PageRank.

The reference manuals regarding node centrality, community detection algorithms and graph generators are the following:
`https://snap.stanford.edu/snappy/doc/reference/centr.html`
`https://snap.stanford.edu/snappy/doc/reference/community.html`
`https://snap.stanford.edu/snappy/doc/reference/generators.html`